IN1010 - våren 2019

Onsdag 16. januar

Java
Objekter og klasser

Stein Gjessing

Institutt for informatikk

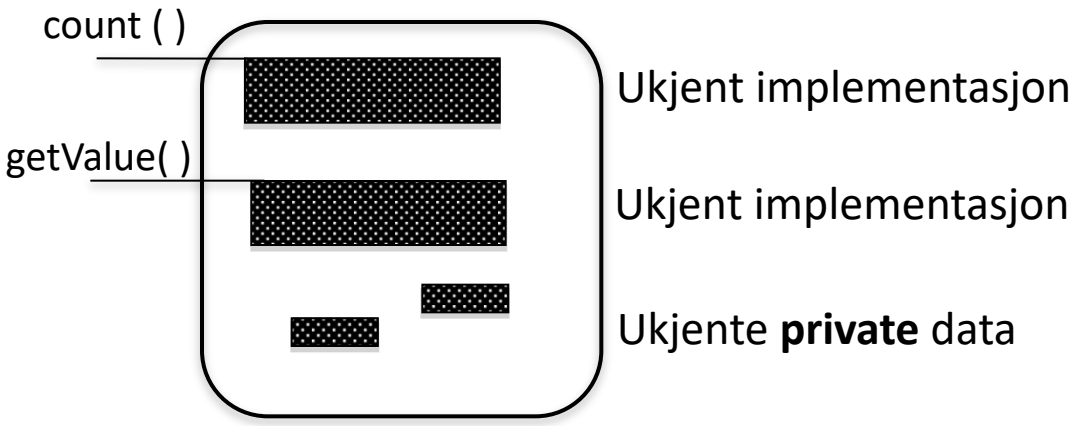
Universitetet i Oslo

IN1010: Objektorientert programmering

- Hva er et objekt ?
- Hva er en klasse ?

- Aller enkleste eksempel (Horstmann kap 8.2):
  - En teller (som f.eks. betjeningen på et fly bruker)
    - Tell én opp
    - Les av telleren nå
  - Starter på null

Et objekt er en sort boks

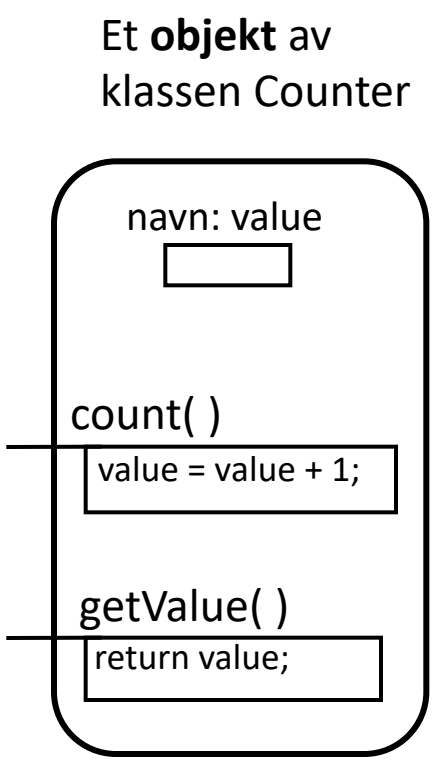


Men den som programmerer (implementerer) klassen må selvfølgelig se inni

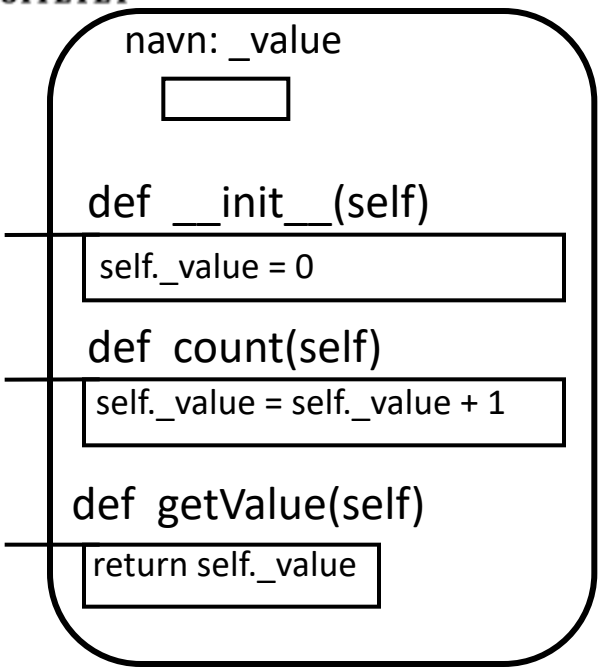
Hva er et objekt ?

Noe som oppstår inne i datamaskinen når vi sier new Counter( );
(hvis vi har deklarert class Counter { . . . })

- Objekter inneholder
  - **Metoder – operasjoner - handlinger**
    - public (som regel)
    - men også private metoder
      - til bruk inne i objektet
  - **Variable og konstanter - "DATA"**
    - av de primitive typene eller pekere
    - som regel skjult for omverdenen – private (innkapsling)

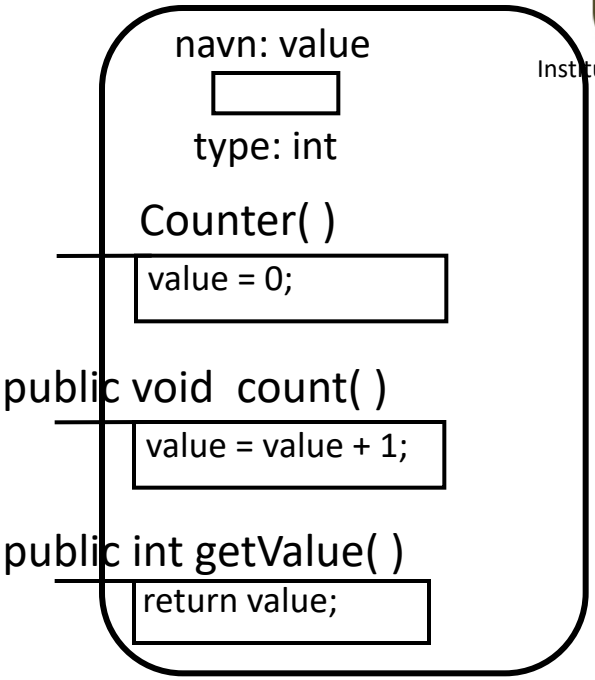


Python



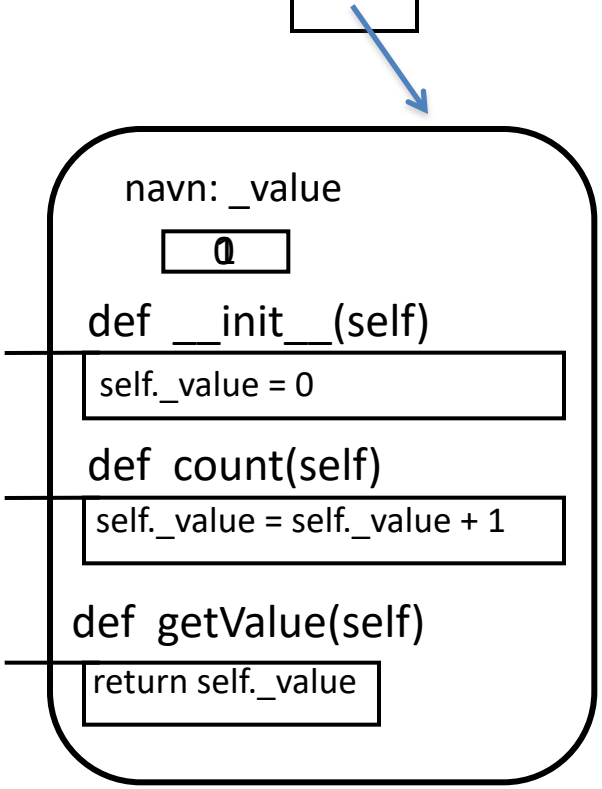

```
class Counter:

    def __init__(self):
        self. _value = 0

    def count(self):
        self._value = self. _value + 1

    def getValue(self):
        return self._value
```

Java




```
class Counter {
    private int value;
    public Counter() {
        value = 0;
    }
    public void count( ) {
        value = value + 1;
    }
    public int getValue( ) {
        return value;
    }
}
```

Python

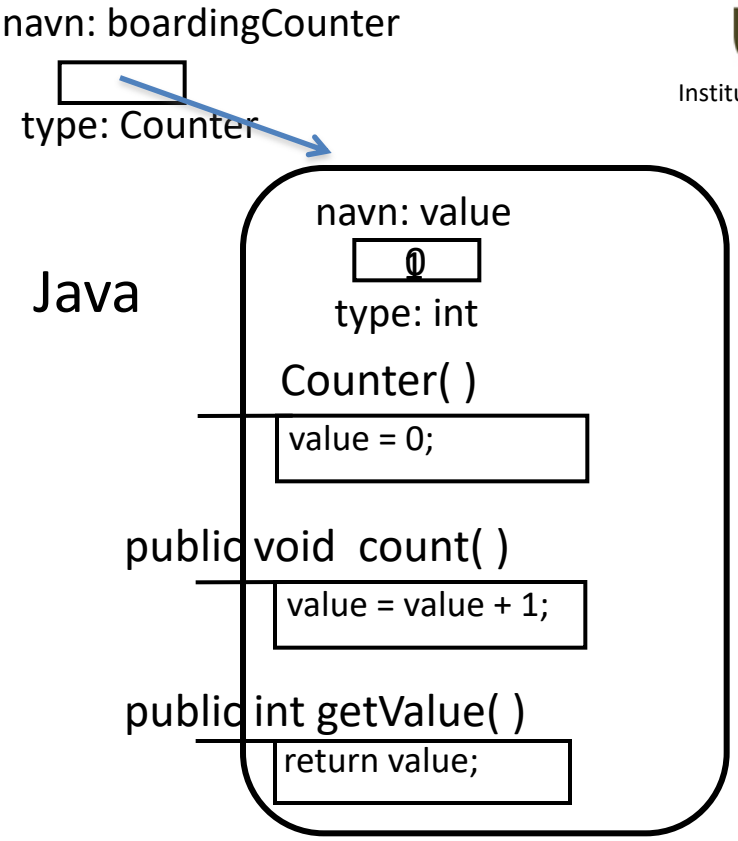



```
boardingCounter = Counter ( );

boardingCounter.count( );

tall = boardingCounter.getValue( );
```

navn: tall

1

Java




```
Counter boardingCounter = new Counter ( );

boardingCounter.count( );

int tall = boardingCounter.getValue( );
```

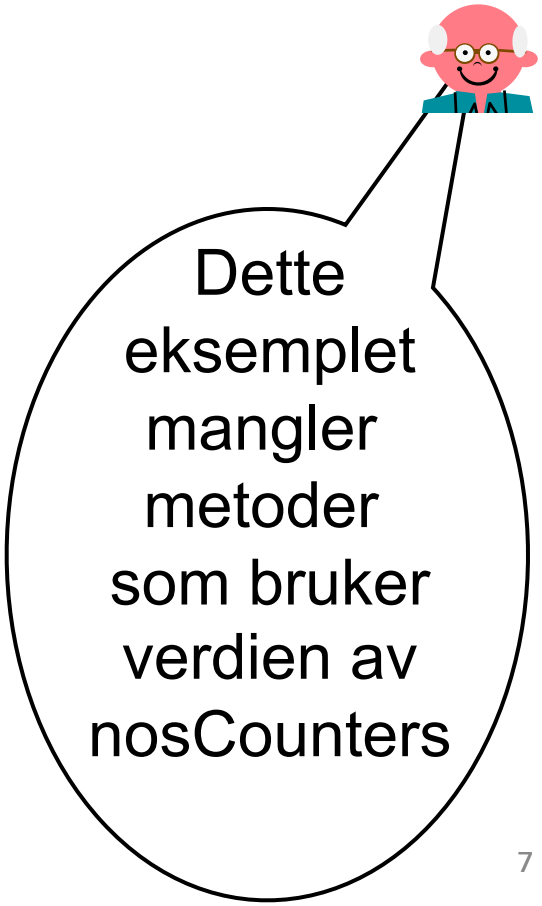
navn: tall

1

type: int

Java: **static**-egenskaper

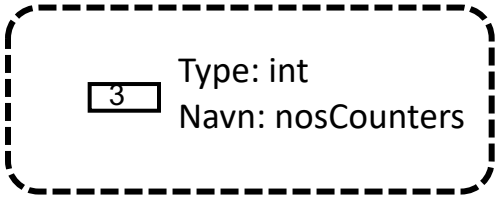
```
class Counter {
    private static int nosCounters =0;
    private int value;
    public Counter() {
        value = 0;
        nosCounters ++;
     }
     public void count( ) {
         value = value + 1;
      }
     public int getValue( ) {
         return value;
     }
}
```



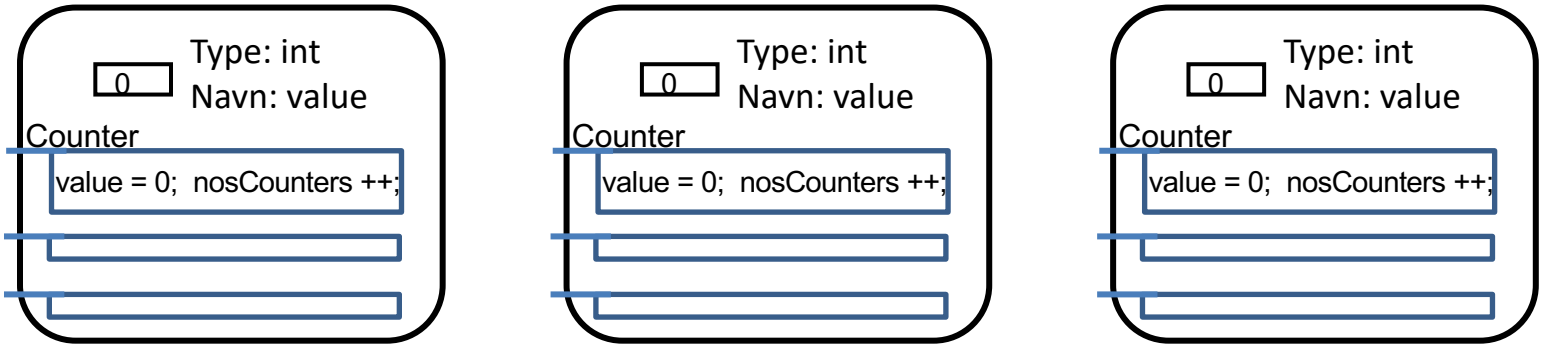
static-egenskaper


```
class Counter {
    private static int nosCounters =0;
    private int value;
    public Counter() {
        value = 0;
        nosCounters ++;
    }
    public void count( ) {
        value = value + 1;
    }
    public int getValue( ) {
        return value;
    }
}
```

Når programmet starter lages en *klasse-datastruktur* av "static"-egenskapene:



Etter f.eks. 3 kall på new Counter() har vi 3 objekter (*objekt-datastruktur*):



De tegningene vi har sett hittil av variable, metoder, objekter og klassedatastrukturer kaller vi til sammen Java **datastrukturer**

Om å løse problemer ved hjelp av datastrukturer

- Når vi skal løse et problem må vi tenke oss en **datastruktur** som løser problemet
- Selve løsningen av problemet er manipulering av denne datastrukturen (en **algoritme**)
- Når du skal løse et problem:
  - Tenk og tegn først datastrukturen
  - Deretter kan du skrive koden (algoritmen) som lager og manipulerer datastrukturen og løser problemet

Mer om (Java) datastrukturer

- Et program oppretter en datastruktur som programmets instruksjoner manipulerer
- Vi må ha et mentalt bilde av dette
- Vi må kunne kommunisere dette bilde til andre som vi programmerer sammen med
- Da tegner vi datastrukturen

Hvor nøyaktig skal jeg tegne Java datastrukturer ?

- Svar:
- Så nøyaktig som det er nødvendig for at du selv eller dem du samarbeider med skal skjønne hva som skjer med datastrukturen når programmet (algoritmen) utføres

- Du må gjerne tegne det på en annen måte enn slik vi gjør i IN1010, men da er det ikke sikkert vi andre skjønner deg

Et litt større eksempel

- Du har en venn som er bruktbilselger, og du skal hjelpe ham med å lage et program for å holde orden på hvor mange som er interessert i de enkelte bilene han har til salgs.
- Først tegner du to bil-objekter, så lager du et program som lager og bruker disse to bil-objektene. Dette programmet skal du senere utvide . . .
- Etter at du først tegnet datastrukturen og så programmerte en stund kom du fram til programmet på neste side
- Hvordan tenkte du?
- Hvordan virker dette programmet?

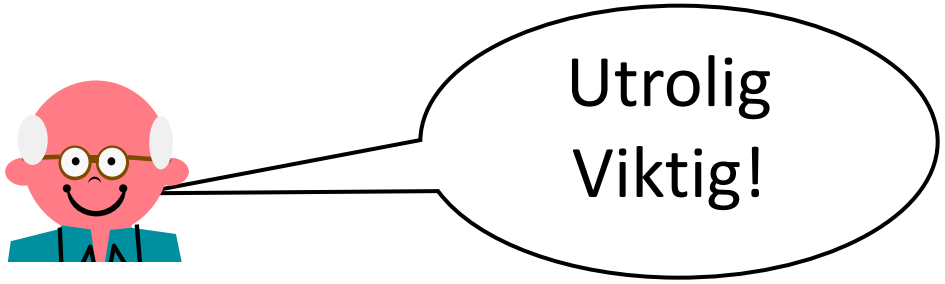
Program for salg av biler.

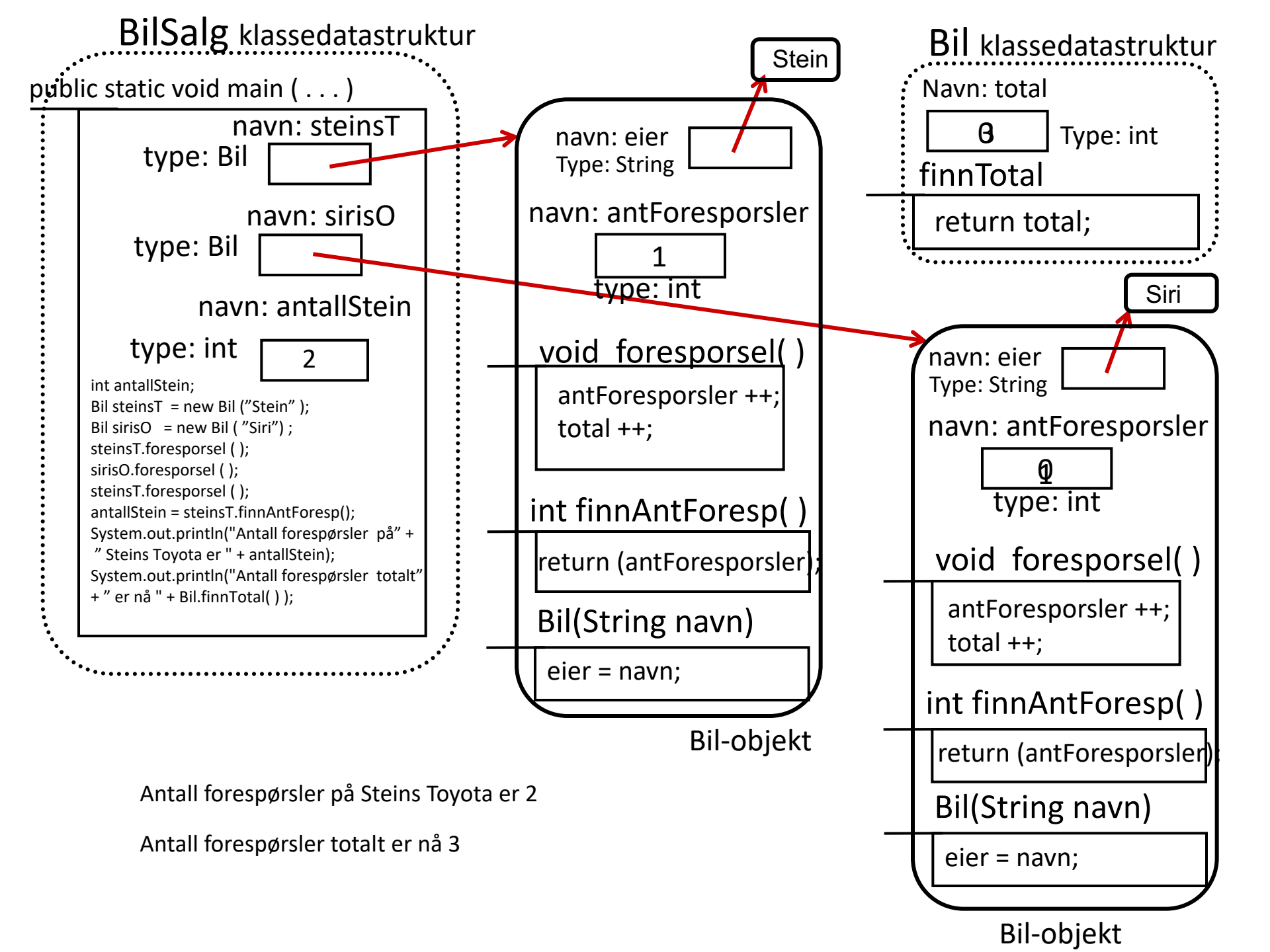
```
public class BilSalg{
   public static void main (String [ ] args) {
      int antallStein;
      Bil steinsT  = new Bil (”Stein”);
      Bil sirisO   = new Bil (”Siri”) ;
      steinsT.foresporsel ( );
      sirisO.foresporsel ( );
      steinsT.foresporsel ( );
      antallStein = steinsT.finnAntForesp();
      System.out.println("Antall forespørsler  på” +
            ” Steins Toyota er " + antallStein);
      System.out.println("Antall forespørsler  totalt” +
            ” er nå " + Bil.finnTotal( ) );
   }
}
```

```
class Bil {
   private static int total = 0;
   private String eier;
   private int antForesporsler = 0;

   public Bil (String  navn) {
         eier = navn;
   }
   public static int finnTotal ( ) {
         return total;
    }
    public void foresporsel ( ) {
         antForesporsler ++;
         total ++;
    }
   public int finnAntForesp ( ) {
          return  antForesporsler;
    }
}
```

Vi skiller mellom

- Klasse-deklarasjonen i programteksten. Den er et mønster som brukes både når klassedatastrukturen lages (i det programmet starter opp) og senere når nye objekter lages.
- Klasse-datastrukturen, dvs. den (statiske) datastrukturen som lages i det programmet starter.
- Objekt-datastrukturen (også kalt klasse-instanser, klasse-objekter eller bare objekter) som lages hver gang vi sier new.





Antall forespørsler på Steins Toyota er 2

Antall forespørsler totalt er nå 3